Data Structure Recitation

Binary Search, Binary Search Tree

Qi Feng

Department of Computer Science, Courant Institute of Mathematical Sciences
qf264@nyu.edu

October 26, 2016

Midterm

- Recursive methods.

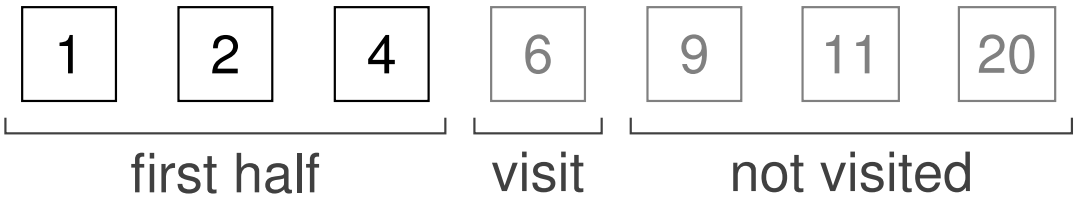
Definition

In computer science, binary search, also known as half-interval search or logarithmic search, is a search algorithm that finds the position of a target value within a sorted array.[1]

[1] https://en.wikipedia.org/wiki/Binary_search_algorithm

Prerequisite and time complexity

- SORTED array.
- $O(\log n)$.

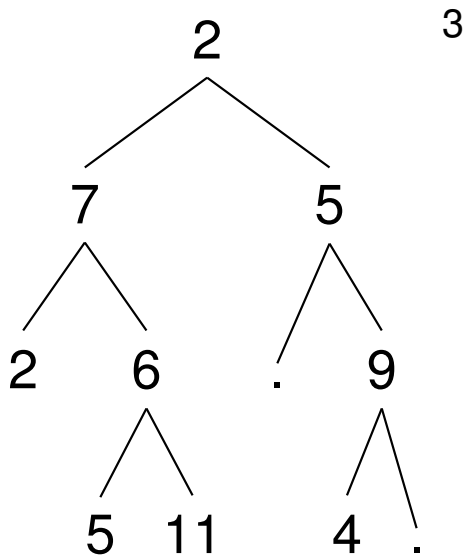

1
2
4
6
9
11
20
first half
visit
not visited

Binary Tree

In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.[2]

[2] https://en.wikipedia.org/wiki/Binary_tree

Example of a binary tree



[3]

[3]"." stands for empty node.

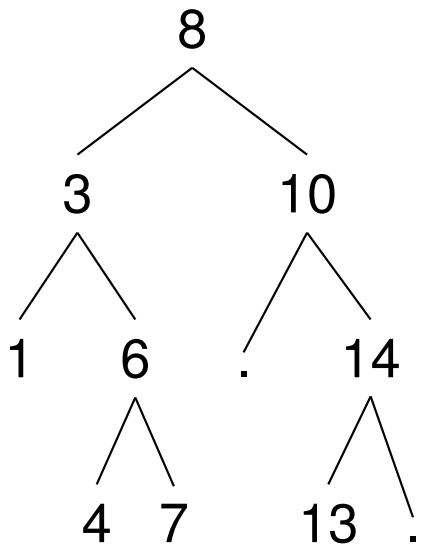
Binary Search Tree

In computer science, binary search trees (BST), sometimes called ordered or sorted binary trees, are a particular type of containers: data structures that store "items" (such as numbers, names etc.) in memory.[4]
Binary search trees keep their keys in sorted order.

- MAX(left subtree) is less than the parent.
- MAX(right subtree is larger than the parent.

[4] https://en.wikipedia.org/wiki/Binary_search _tree

Example of a BST



Features of BST

fast

- lookup
- addition
- removal

Time complexity

	Average	Worst case
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Table: Time complexity of three operations using BST.

Time complexity using a naive array

	Average	Worst case
Search	$O(n)$	$O(n)$
Insert	$O(1)$	$O(1)$
Delete	$O(n)$	$O(n)$

Table: Time complexity of three operations using a naive array.

Lookup

```
1  def search_iteratively(key, node):
2      current_node = node
3      while current_node != null:
4          if key == current_node.key:
5              return current_node
6          else if key < current_node.key:
7              current_node = current_node.left
8          else:  # key > current_node.key:
9              current_node = current_node.right
10     return null
```

Insert

```
1  Node insert(Node root, int key, int value) {
2    if (root == null)
3      root = new Node(key, value);
4    else if (key < root.key)
5      root.left = insert(root.left, key, value);
6    else  // key >= root->key
7      root.right = insert(root.right, key, value);
8    return root;
9  }
```

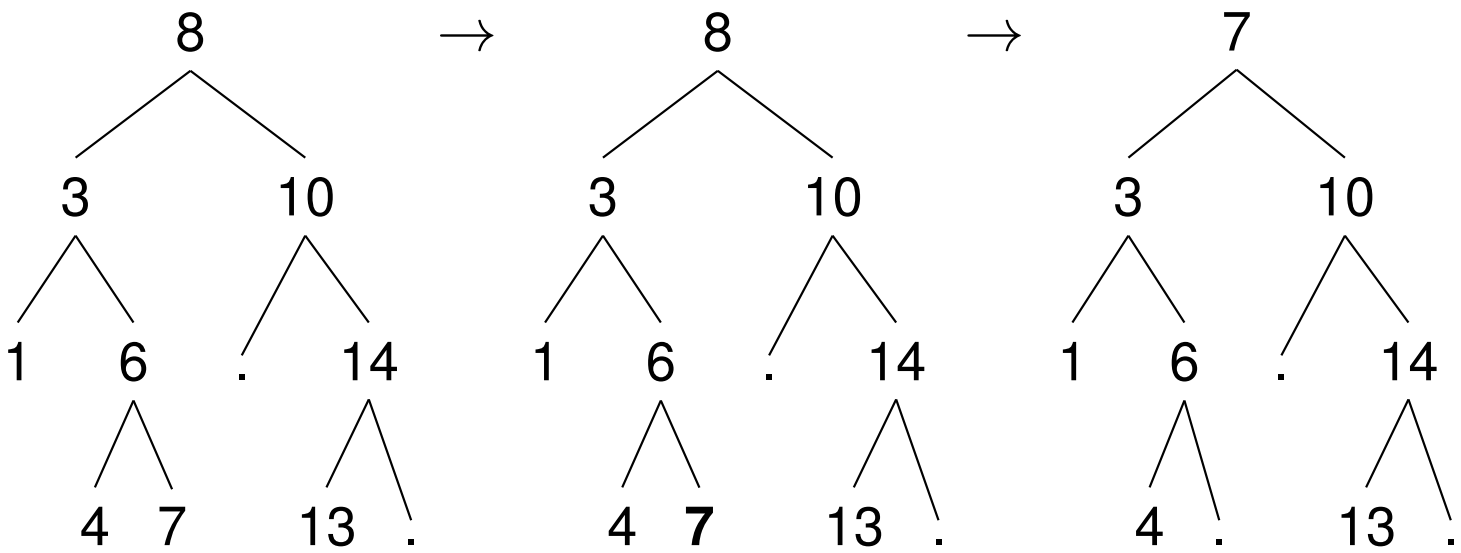
Cut operation

- Cutting a node with no children: simply remove the node from the tree.
- Cutting a node with one child: remove the node and replace it with its child.

Deletion

- Find the node.
- Delete a node with at most one child: cut the tree.
- Delete a node with two children: call the node to be deleted *N*. Do not delete *N*. Instead, choose either its in-order successor node or its in-order predecessor node, *R*. Copy the value of *R* to *N*, then cut *R*.

Delete 8 from BST



Question

Question?